Exercices

Réalisez chacun des exercices ci-dessous. Pour chacun, proposez aussi un programme de test démontrant le bon fonctionnement de votre solution.

Nous présumerons pour le moment que les paramètres utilisés sont tous valides, et éviterons ainsi tout traitement d'erreur.

Une fois que vous serez satisfait(e)s de votre solution pour chaque problème, conservez-la précieusement, puisque nous les raffinerons au cours de la session.

EX00 – On vous donne la fonction `transformer_majuscules()` suivante :

```
#include <string>
#include <locale>
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;
string transformer_majuscules(string originale) {
   string resultat;
   for (string::size_type i = 0; i < originale.size(); i++) {
      char c;
      c = toupper(originale[i], locale::classic());
      resultat = resultat + c;
   }
   return resultat;
}
int main() {
   const int TAILLE = 5'000'000;
   string texte(TAILLE, 'a');
   string resultat;
   auto avant = system_clock::now();
   resultat = transformer_majuscules(texte);
   auto apres = system_clock::now();
   cout << "Transformer en majuscules v1 " << TAILLE << " caractères: "
        << duration_cast<milliseconds>(apres- avant).count() << " ms." << endl;
}
```

Pouvez-vous en améliorer la vitesse d'exécution sans que cela n'ait d'impact sur le code appelant?

EX01 – On vous donne la fonction `trier_texte()` suivante :

```
#include <string>
#include <random>
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;
string trier_texte(string originale) {
   string resultat, avant_milieu, apres_milieu;
   string::size_type milieu;
   switch (originale.size()) {
   case 2:
      if (originale[0] < originale[1]) {
         resultat += originale[0];
         resultat += originale[1];
      } else {
         resultat += originale[1];
         resultat += originale[0];
      }
      break;
   case 1:
   case 0:
      resultat = originale;
      break;
   default:
      milieu = originale.size() / 2;
      avant_milieu = trier_texte(originale.substr(0, milieu));
      apres_milieu = trier_texte(originale.substr(milieu));
      string::size_type av = 0, ap = 0;
      while (av < avant_milieu.size() && ap < apres_milieu.size()) {
         if (avant_milieu[av] < apres_milieu[ap]) {
            resultat += avant_milieu[av];
            av++;
         } else {
            resultat += apres_milieu[ap];
            ap++;
         }
      }
      if (av < avant_milieu.size()) {
         resultat += avant_milieu.substr(av);
      } else if (ap < apres_milieu.size()) {
         resultat += apres_milieu.substr(ap);
      }
   }
   return resultat;
}
```

```
string generer_texte(int taille) {
   random_device rd;
   mt19937 prng { rd() };
   uniform_int_distribution<int> de{ 0, static_cast<int>('z'-'a') };
   string temp;
   for (int i = 0; i < taille; i++)
      temp += de(prng) + 'a';
   return temp;
}
int main() {
   const int TAILLE = 100'000;
   auto texte = generer_texte(TAILLE);
   auto avant = system_clock::now();
   auto texteTrie = trier_texte(texte);
   auto apres = system_clock::now();
   cout << "Avant: " << texte << endl
        << "Après: " << texteTrie << endl
        << "Tri de " << TAILLE << " caractères: "
        << duration_cast<milliseconds>(apres- avant).count() << " ms." << endl;
}
```

Pouvez-vous en améliorer la vitesse d'exécution sans que cela n'ait d'impact sur le code appelant?

EX02 – Complétez les classes `Noeud` et `Liste` suivantes en leur ajoutant les opérateurs, les constructeurs et les destructeurs requis pour que le programme en faisant usage fonctionne, et pour que l'utilisation de cette classe soit sécuritaire. Ayez en tête l'objectif d'avoir un code à la fois élégant et efficace (vous ne pouvez pas modifier les méthodes existantes, même si je sais autant que vous qu'on peut faire mieux) :

```
struct Noeud {
   Noeud *succ;
   int val;
   // AJOUTEZ ICI
};
class ListeEntiers {
   Noeud *tete;
public:
   class ListeVide { };
   // AJOUTEZ ICI
   bool est_vide() const noexcept {
      return tete == nullptr;
   }
   void ajouter(int val) {
      if (est_vide())
         tete = new Noeud{val};
      else {
         auto p = tete;
         while (p->succ) p = p->succ;
         p->succ = new Noeud{val};
      }
   }
   int extraire() {
      if (est_vide()) throw ListeVide{};
      auto p = tete;
      tete = tete->succ;
      const int val = p->val;
      delete p;
      return val;
   }
   int taille() const noexcept {
      int n = 0;
      for (auto p = tete; p; p = p->succ)
         ++n;
      return n;
   }
   void inverser() {
      Noeud *nouvelle_tete = {};
      while(tete) {
         auto p = new Noeud{*tete};
         p->succ = nouvelle_tete;
```

```
            nouvelle_tete = p;
            p = tete;
            tete = tete->succ;
            delete p;
         }
         tete = nouvelle_tete;
      }
};
#include <iostream>
int main() {
   using namespace std;
   const int NB_ENTIERS = 10;
   ListeEntiers lst;
   for (int i = 0; i < NB_ENTIERS; i++)
      lst.ajouter(i + 1);
   ListeEntiers lstInv = lst;
   lstInv.inverser();
   const int NB_ELEMENTS = lstInv.taille();
   for (int i = 0; i < NB_ELEMENTS; i++)
      cout << lstInv.extraire() << " ";
   cout << endl;
}
```

EX03 – Dans le programme ci-dessus, remplacez la répétitive for par une boucle qui extrait et affiche un élément de la liste `lstInv` et ce, *tant qu'aucune exception n'aura été levée*.

EX04 – Dans le programme ci-dessus, apportez des améliorations à l'implémentation de la classe `ListeEntiers` sans en changer l'interface, pour que les méthodes soient aussi efficaces que possible. Il y a *beaucoup* de choses que vous pouvez faire…

EX05 – Soit le programme suivant, dont la compilation génère des erreurs et des avertissements. Veuillez y insérer les conversions explicites de types les plus appropriées pour qu'il compile sans erreur et sans avertissement. Ne procédez à aucune autre modification qu'à l'insertion de conversions explicites de types.

```
#include <string>
using std::string;
void encodage_full_cool(unsigned char *texte) {
   for(;*texte;++texte) {
      const int EVENTAIL_LETTRES = ('z' - 'a' + 1);
      const int ROTATION = EVENTAIL_LETTRES / 2; // algo ROT 13
      const unsigned char PLANCHER =
         (*texte >= 'a' && *texte <= 'z')? 'a' : 'A';
      *texte = (*texte - PLANCHER + ROTATION) % EVENTAIL_LETTRES + PLANCHER;
   }
}
#include <iostream>
#include <vector>
#include <algorithm>
int main() {
   using namespace std;
   const auto MESSAGE = "Amusez-vous bien!"s;
   vector<unsigned char> txt(begin(MESSAGE), end(MESSAGE));
   txt.push_back(0);
   cout << "Taille du texte brut: " << txt.size()<< endl;
   cout << "Avant: " << txt.data() << endl;
   encodage_full_cool(&txt[0]);
   cout << "Après encodage: " << txt.data() << endl;
   encodage_full_cool(&txt[0]);
   cout << "Après décodage: " << txt.data() << endl;
}
```

Pour vous amuser

Ce programme est une forme incomplète du célèbre algorithme ROT13, décrit entre autres sur http://en.wikipedia.org/wiki/ROT13. Il ne gère bien que les caractères alphabétiques. Pouvez-vous l'améliorer de manière à ce qu'il gère la totalité des caractères alphanumériques?